

[illegible]

```
1  /*
2                                     EDIT: HE2004
3  ****
4  **
5  ** COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
6  ** DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
7  ** ALL RIGHTS RESERVED.
8  **
9  ** THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
10 ** ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
11 ** INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
12 ** COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
13 ** OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
14 ** TRANSFERRED.
15 **
16 ** THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
17 ** AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
18 ** CORPORATION.
19 **
20 ** DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
21 ** SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
22 **
23 ****
24
25 facility:      VAX-11 PL/I Runtime Library.
26
27 abstract:      This routine is called to process the environment attributes
28                 for the PL/I open service.
29
30 author:        C. Spitz
31
32 date:          23-Jan-1980
33
34 Modifications:
35   V1.4-02:      Bill Matthews   28-Sep-1981
36
37                 Fix to not maximize versions ever when an explicit version
38                 number is specified.
39
40   V1.4-03:      Bill Matthews   08-Oct-1981
41
42                 Fix coding of protection utility to not rely on short circuit
43                 boolean optimization for correct execution of the program.
44
45   V2.0-04:      Hisham Elbasha  11-NOV-1982
46
47                 make the upi bit independent of the bio bit for shared_read
48                 and shared_write.
49
50 */
51
52 Local Commentary:
53   The environment options for a file may be specified on the DECLARE
54   statement for the file, on the OPEN statement, or on the CLOSE
55   statement. The environment options are represented as a list of
56   elements, where each element is represented by its type code, and
```



```
56 : its value. The type code is one byte long; valid environments have
57 : values of 1 through num_envir_opts. The value of 0 is used to des-
58 : ignate the end of the environment list. Each environment option has
59 : a parameter, whose interpretation is dependant upon the option. The
60 : parameters data types are:
61 :     immediate bit - represented as 1 byte, low bit = value
62 :     immediate value - represented as 1 longword
63 :     immediate character - represented as n bytes. the first
64 :         2 bytes are the total length of the character
65 :         string, the second 2 bytes are the current length
66 :         of the character sting, and the remaining n-4 bytes
67 :         are storage for the total length of the string. Note
68 :         that both lengths do not include the length fields.
69 :     address - represented as a 4 byte absolute address.
70 :     quad value - represented as a 4 byte absolute address. */
71 :
72 : pli$envir: procedure(fcbpt,openv,open_blk) options(ident('1-004'))
73 :     returns(fixed bin(31));
74 :
75 : /* parameter declarations */
76 : dcl      fcbpt          pointer, /* pointer to file control block */
77 :          openv          pointer, /* pointer to open environment */
78 :          open_blk       pointer; /* pointer to open block */
79 :
80 : /* the following is a template for the macro open block */
81 : dcl      1 opn          based(open_blk),
82 :          2 status(0:31) bit,
83 :          2 create_date(0:1) fixed bin(31),
84 :          2 expire_date(0:1) fixed bin(31),
85 :          2 file_id_to_pt pointer,
86 :          2 fixed_control_to_pt pointer,
87 :          2 prot(0:15) bit,
88 :          2 own_group     fixed bin(15),
89 :          2 own_mem       fixed bin(15);
90 :
91 : /* bit offsets for status */
92 : %replace create_dat by 0;
93 : %replace expire_dat by 1;
94 : %replace fileid_to by 2;
95 : %replace fixedctl_to by 3;
96 : %replace protect by 4;
97 : %replace uic by 5;
98 : %replace close by 6;
99 :
100 : /* bit offsets for protection */
101 : %replace no_read by 0;
102 : %replace no_write by 1;
103 : %replace no_execute by 2;
104 : %replace no_delete by 3;
105 : %replace system_prot by 0;
106 : %replace owner_prot by 4;
107 : %replace group_prot by 8;
108 : %replace world_prot by 12;
109 :
110 : /* general constants */
111 : %replace true by '1'b;
112 : %replace false by '0'b;
113 :
114 : /* global declarations */
```

PLISSENVIR
1-004

D 15
16-SEP-1984 02:29:35
6-SEP-1984 11:37:37

VAX-11 PL/I X2.1-273 Page 3
ISK\$VMSMASTER:[PLIRTL.SRC]PLIENVIR.PLI;1 (1)

113	1	%include envcodes; /* define environment codes and types */
171	1	%include filedef; /* define file control block, fab, rab, nam*/
408	1	

```
409 : 1 /* local data - static */
410 : 1 /* the following table contains the parameter type for each environment option*/
411 : 1 %replace bittyp by 0:
412 : 1 %replace longtyp by 1:
413 : 1 %replace quadtyp by 2:
414 : 1 %replace stringtyp by 3:
415 : 1 %replace addrtyp by 4:
416 : 1 dcl env_type(num_envir_opts) fixed bin(7) static readonly
417 : 1 init( bittyp, /* append */
418 : 1 bittyp, /* batch */
419 : 1 bittyp, /* block_boundry */
420 : 1 bittyp, /* block_io */
421 : 1 longtyp, /* block_size */
422 : 1 longtyp, /* bucket_size */
423 : 1 bittyp, /* carriage */
424 : 1 bittyp, /* contiguous */
425 : 1 bittyp, /* contiguous_best_try */
426 : 1 quadtyp, /* creation_date */
427 : 1 bittyp, /* current_position */
428 : 1 stringtyp, /* default_file_name */
429 : 1 bittyp, /* deferred_write */
430 : 1 bittyp, /* delete */
431 : 1 quadtyp, /* expiration_date */
432 : 1 longtyp, /* extension_size */
433 : 1 addrtyp, /* file_id */
434 : 1 addrtyp, /* file_id_to */
435 : 1 longtyp, /* file_size */
436 : 1 longtyp, /* fixed_control_size */
437 : 1 addrtyp, /* fixed_control_size_to */
438 : 1 bittyp, /* fixed_length_records */
439 : 1 stringtyp, /* group_protection */
440 : 1 bittyp, /* ignore_line_marks */
441 : 1 bittyp, /* indexed */
442 : 1 bittyp, /* indexed_fill */
443 : 1 longtyp, /* index_number */
444 : 1 longtyp, /* max_record_number */
445 : 1 longtyp, /* max_record_size */
446 : 1 longtyp, /* multiblock_count */
447 : 1 longtyp, /* multibuffer_count */
448 : 1 bittyp, /* no_share */
449 : 1 longtyp, /* owner_group */
450 : 1 longtyp, /* owner_member */
451 : 1 stringtyp, /* owner_protection */
452 : 1 bittyp, /* printer */
453 : 1 bittyp, /* read_ahead */
454 : 1 bittyp, /* read_check */
455 : 1 bittyp, /* record_id_access */
456 : 1 longtyp, /* retrieve_pointers */
457 : 1 bittyp, /* rewind_close */
458 : 1 bittyp, /* rewind_open */
459 : 1 bittyp, /* scalar_varying */
460 : 1 bittyp, /* shared_read */
461 : 1 bittyp, /* shared_write */
462 : 1 bittyp, /* spool */
463 : 1 bittyp, /* supersede */
464 : 1 stringtyp, /* system_protection */
```


PLISSENVIR
1-004

F 15
16-SEP-1984 02:29:35
6-SEP-1984 11:37:37

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[PLIRTL.SRC]PLIENVIR.PLI;1 (2) Page 5

```
465      1      bittyp,      /* temporary */
466      1      bittyp,      /* truncate */
467      1      stringtyp,    /* world_protection */
468      1      bittyp,      /* write_behind */
469      1      bittyp);      /* write_check */
470
471      1      dcl      1 end_opt      static readonly, /* end of environment list */
472      1                  2 env_number      fixed bin(7) init(UNUSED_ENVIR_OPT);
473
474      1      dcl      default_name      char(4) static readonly init('.DAT');
475      1
```

```
476 : 1      /* local data - automatic */
477 : 1      dcl      fcb      pointer, /* local pointer to fcb (unaliased) */
478 : 1          declared_environment pointer,
479 : 1          current_env_number   fixed bin(7),
480 : 1          next_specified_env_number fixed bin(7),
481 : 1          longtemp             fixed bin(31),
482 : 1          point               pointer, /* utility pointer */
483 : 1          error_code           fixed bin(31),
484 : 1          carriage_specified_false bit aligned, /* true if carriage was specified
485 : 1                                          as '0'b */
486 : 1          specified           bit aligned; /* true if current_env_number was
487 : 1                                          specified in an environment list */
488 : 1
489 : 1      /* the following are used to compare the declared and open environments, to
490 : 1      ensure that they are the same. THEY ARE NOT AVAILABLE FOR USE AS TEMPS. */
491 : 1      dcl      bitval(0:1)      bit aligned,
492 : 1          addrval(0:1)          pointer,
493 : 1          longval(0:1)         fixed bin(31),
494 : 1          quadval(0:1,0:1)     fixed bin(31);
495 : 1      /*
496 : 1
```



```
497 : 1 /* based declarations */
498 : 1 /* the following declarations are templates for the various types of environment
499 : 1 options. there is one template for each parameter type. */
500 : 1 dcl 1 optbit based,
501 : 1 2 env_number fixed bin(7),
502 : 1 2 bitf bit,
503 : 1 2 bitext(7) bit,
504 : 1 2 bitnext fixed bin(7);
505 : 1 dcl 1 optlong based,
506 : 1 2 env_number fixed bin(7),
507 : 1 2 long fixed bin(31),
508 : 1 2 longnext fixed bin(7);
509 : 1 dcl 1 optaddr based,
510 : 1 2 env_number fixed bin(7),
511 : 1 2 address pointer,
512 : 1 2 addrnext fixed bin(7);
513 : 1 dcl 1 optstring based,
514 : 1 2 env_number fixed bin(7),
515 : 1 2 maxsize fixed bin(15),
516 : 1 2 string char(128) var;
517 : 1 dcl 1 optstringnext based,
518 : 1 2 env_number fixed bin(7),
519 : 1 2 maxsize fixed bin(15),
520 : 1 2 cursize fixed bin(15),
521 : 1 2 stringnext(0:128) fixed bin(7);
522 : 1
523 : 1 /* the following are templates for moving values around */
524 : 1 dcl value fixed bin(31) based;
525 : 1 dcl qvalue(0:1) fixed bin(31) based;
526 : 1 dcl byte fixed bin(7) based(addr(longval));
527 : 1 dcl word fixed bin(15) based(addr(longval));
528 : 1 dcl fileid char(22) based(addrval(0));
529 : 1 dcl bytetemp fixed bin(7) based(addr(longtemp));
530 : 1 dcl wordtemp fixed bin(15) based(addr(longtemp));
531 : 1 dcl buflen fixed bin(15) based(
532 : 1 addr(fcb->file_constant.buffer_end));
533 : 1 dcl stringtemp char(128) var based;
534 : 1 dcl 1 s based,
535 : 1 2 stringlen fixed bin(15),
536 : 1 2 stringval char(128);
537 : 1
```

```
538 : 1 /* declarations of error messages and error routines */
539 : 1 dcl      pli$io_error      entry(fixed bin(31) value,
540 : 1          fixed bin(31) value, pointer value);
541 : 1 dcl      pli$_undfile      globalref fixed bin(31) value;
542 : 1 dcl      pli$_envparm      globalref fixed bin(31) value;
543 : 1 dcl      pli$_invdfnam      globalref fixed bin(31) value;
544 : 1 dcl      pli$_conappsup      globalref fixed bin(31) value;
545 : 1 dcl      pli$_conblokio      globalref fixed bin(31) value;
546 : 1 dcl      pli$_invrtvptr      globalref fixed bin(31) value;
547 : 1 dcl      pli$_noshare      globalref fixed bin(31) value;
548 : 1 dcl      pli$_invprot      globalref fixed bin(31) value;
549 : 1 dcl      pli$_invmltblk      globalref fixed bin(31) value;
550 : 1 dcl      pli$_invmltbuf      globalref fixed bin(31) value;
551 : 1 dcl      pli$_confixlen      globalref fixed bin(31) value;
552 : 1 dcl      pli$_invindnum      globalref fixed bin(31) value;
553 : 1 dcl      pli$_invblksiz      globalref fixed bin(31) value;
554 : 1 dcl      pli$_invbktsiz      globalref fixed bin(31) value;
555 : 1 dcl      pli$_invextsiz      globalref fixed bin(31) value;
556 : 1 dcl      pli$_invfxcsiz      globalref fixed bin(31) value;
557 : 1 dcl      pli$_conenvopt      globalref fixed bin(31) value;
558 : 1 dcl      pli$_conprintcr      globalref fixed bin(31) value;
559 : 1 dcl      pli$_invowngrp      globalref fixed bin(31) value;
560 : 1 dcl      pli$_invownmem      globalref fixed bin(31) value;
561 : 1 dcl      pli$_conprtfrm      globalref fixed bin(31) value;
562 : 1 dcl      pli$_creindex      globalref fixed bin(31) value;
563 : 1 dcl      pli$_invmaxrec      globalref fixed bin(31) value;
564 : 1
```

```

565 : 1 /* initialization */
566 : 1 /* define general error condition handler */
567 : 1 on anycondition begin;
568 : 2     error_code = pli$envparm;
569 : 2     goto opt_error;
570 : 2 end;
571 : 1
572 : 1 fcb = fcbpt; /* copy fcb pointer to local storage */
573 : 1 declared_environment = addr(fcb -> fcb_end); /* point to declared environment */
574 : 1 if openv = null()
575 : 1     then openv = addr(end_opt);
576 : 1 if fcb -> fcb_end = 0 ! opn.status(close)
577 : 1     then declared_environment = addr(end_opt);
578 : 1 next_specified_env_number = 0;
579 : 1

```



```
580 : 1      /* main loop */
581 : 1      do current_env_number = 0 to num_envir_opts;
582 : 2          specified = (next_specified_env_number = current_env_number);
583 : 3          if opn.status(close)
584 : 4              then do;
585 : 5                  if current_env_number = batch !
586 : 6                      current_env_number = delete !
587 : 7                      current_env_number = rewind_close !
588 : 8                      current_env_number = spool !
589 : 9                      current_env_number = truncate
590 : 10                     then goto opt(current_env_number);
591 : 11                     end;
592 : 12                 else goto opt(current_env_number);
593 : 13             goto next_opt;
594 : 14
595 : 15      /* error routine */
596 : 16
597 : 17      opt_error:
598 : 18          revert anycondition;
599 : 19          call plisio_error(plis_undfile,error_code,fcb);
600 : 20          return(plis_undfile);
601 : 21
```

```
602 opt(0):  
603     goto next_opt;  
604 opt(append):  
605     if specified & bitval(0)  
606         then do;  
607         fcb -> attr(atr_v_app) = true;  
608         fcb -> fab$l_fop(fab$u_mxv) = false;  
609         fcb -> fab$l_fop(fab$u_cif) = true;  
610         fcb -> fab$l_fop(fab$u_sup) = false;  
611         fcb -> fab$l_fop(fab$u_nef) = false;  
612         fcb -> rab$l_rop(rab$u_eof) = true;  
613         end;  
614     else fcb -> attr(atr_v_app) = false;  
615     goto next_opt;  
616  
617  
618 opt(batch):  
619     fcb -> fab$l_fop(fab$u_scf) = specified & bitval(0);  
620     goto next_opt;  
621  
622  
623 opt(block_boundry):  
624     fcb -> fab$b_rat(fab$u_blk) = specified & bitval(0);  
625     goto next_opt;  
626  
627  
628 opt(block_io):  
629     if specified & bitval(0) : fcb -> attr(atr_v_blockio)  
630         then do;  
631         if fcb -> fab$b_rat(fab$u_blk)  
632             : fcb -> attr(atr_v_stream)  
633             then do;  
634                 error_code = pli$conblkio;  
635                 goto opt_error;  
636             end;  
637         fcb -> fab$b_fac(fab$u_bio) = true;  
638         fcb -> fab$b_rfm = fab$c_udf;  
639         end;  
640     else fcb -> fab$b_fac(fab$u_bio) = false;  
641     fcb -> fab$b_shr(fab$u_upi) = false;  
642     goto next_opt;  
643  
644  
645 opt(block_size):  
646     if specified  
647         then do;  
648         if longval(0) < 0 : longval(0) > 65535  
649             then do;  
650                 error_code = pli$invblksiz;  
651                 goto opt_error;  
652             end;  
653         fcb -> fab$w_bls = word;  
654         end;  
655     else fcb -> fab$w_bls = 0;  
656     goto next_opt;  
657
```

```
opt(bucket_size):
  if specified
  then do;
    if longval(0) < 0 : longval(0) > 32
    then do;
      error_code = pli$_invbktsiz;
      goto opt_error;
    end;
    fcb -> fab$b_bks = byte;
  end;
  else fcb -> fab$b_bks = 0;
  goto next_opt;

opt(carriage):
  if specified & bitval(0)
  then do;
    if fcb -> attr(atr_v_print)
    then do;
      error_code = pli$_conprinter;
      goto opt_error;
    end;
    if fcb -> fab$b_fac(fab$v_bio)
    then do;
      error_code = pli$_conblokio;
      goto opt_error;
    end;
    fcb -> fab$b_rat(fab$v_cr) = true;
  end;
  else do;
    fcb -> fab$b_rat(fab$v_cr) = false;
    carriage_specified_false = specified;
  end;
  fcb -> fab$b_rat(fab$v_ftn) = false;
  goto next_opt;

opt(contiguous):
  fcb -> fab$l_fop(fab$v_ctg) = specified & bitval(0);
  goto next_opt;

opt(contiguous_best_try):
  fcb -> fab$l_fop(fab$v_cbt) = specified & bitval(0);
  goto next_opt;

opt(creation_date):
  if specified
  then do;
    create_date(0) = quadval(0,0);
    create_date(1) = quadval(0,1);
    opn.status(create_dat) = true;
  end;
  goto next_opt;
```



```
715  
716  
717     opt(current_position):  
718         fcb->fab$l_fop(fab$u_pos) = specified & bitval(0);  
719         goto next_opt;  
720  
721     opt(default_file_name):  
722         if specified  
723             then do;  
724                 if addrval(0) -> stringlen > 128  
725                     then do;  
726                         error_code = pli$_invdfnam;  
727                         goto opt_error;  
728                     end;  
729                 fcb->fab$l_dna = addr(addrval(0) -> stringval);  
730                 longtemp = addrval(0) -> stringlen;  
731                 fcb->fab$b_dns = bytetemp;  
732                 end;  
733             else do;  
734                 fcb->fab$l_dna = addr(default_name);  
735                 fcb->fab$b_dns = length(default_name);  
736             end;  
737             goto next_opt;  
738  
739     opt(defered_write):  
740         fcb->fab$l_fop(fab$u_dfw) = specified & bitval(0);  
741         goto next_opt;  
742  
743     opt(delete):  
744         fcb->fab$l_fop(fab$u_dlt) = specified & bitval(0);  
745         goto next_opt;  
746  
747     opt(expiration_date):  
748         if specified  
749             then do;  
750                 expire_date(0) = quadval(0,0);  
751                 expire_date(1) = quadval(0,1);  
752                 opn.status(expire_dat) = true;  
753                 end;  
754             goto next_opt;  
755  
756     opt(extension_size):  
757         if specified  
758             then do;  
759                 if longval(0) < 0 : longval(0) > 65535  
760                     then do;  
761                         error_code = pli$_invextsiz;  
762                         goto opt_error;  
763                     end;  
764                 fcb->fab$w_deq = word;  
765                 end;  
766             else fcb->fab$w_deq = 0;  
767             goto next_opt;  
768  
769  
770  
771
```

```
772  
773  
774 opt(file_id):  
775     If specified  
776         then do;  
777             fcb -> nam$st_dvi = fileid;  
778             fcb -> nam$sw_did = 0;  
779             fcb -> nam$sw_did_seq = 0;  
780             fcb -> nam$sw_did_rvn = 0;  
781             fcb -> fab$l_fop(fab$sv_nam) = true;  
782             end;  
783         else fcb -> fab$l_fop(fab$sv_nam) = false;  
784     goto next_opt;  
785  
786  
787 opt(file_id_to):  
788     If specified  
789         then do;  
790             file_id_to_pt = addrval(0);  
791             opn$status(fileid_to) = true;  
792             end;  
793     goto next_opt;  
794  
795  
796 opt(file_size):  
797     If specified  
798         then fcb -> fab$l_alq = longval(0);  
799         else fcb -> fab$l_alq = 0;  
800     goto next_opt;  
801  
802  
803 opt(fixed_control_size):  
804     if specified  
805         then do;  
806             if fcb -> attr(atr_v_stream) ;  
807                 fcb -> attr(atr_v_update) ;  
808                 fcb -> fab$b_fac(fab$sv_bio) ;  
809                 longval(0) < 0 ; longval(0) > 255  
810             then do;  
811                 error_code = pli$invfxsiz;  
812                 goto opt_error;  
813             end;  
814             fcb -> fab$b_fsz = byte;  
815             fcb -> fab$b_rfm = fab$sc_vfc;  
816             end;  
817         else do;  
818             if fcb -> attr(atr_v_print)  
819                 then do;  
820                 fcb -> fab$b_fsz = 2;  
821                 fcb -> fab$b_rfm = fab$sc_vfc;  
822                 fcb -> fab$b_rat(fab$sv_prn) = true;  
823                 end;  
824             else fcb -> fab$b_fsz = 0;  
825         end;  
826     goto next_opt;  
827  
828
```

```
829 opt(fixed_control_size_to):
830     if specified
831         then do;
832             fixed_control_to_pt = addrval(0);
833             opn.status(fixedctl_to) = true;
834             end;
835     goto next_opt;
836
837 opt(fixed_length_records):
838     if specified & bitval(0)
839         then do;
840         if (fcb -> attr(atr_v_stream) &
841             fcb -> attr(atr_v_output)) ;
842             (fcb -> fab$b_rfm = fab$c_vfc) ;
843             (fcb -> fab$b_fac(fab$v_bio))
844         then do;
845             error_code = pli$confixlen;
846             goto opt_error;
847             end;
848         fcb -> fab$b_rfm = fab$c_fix;
849         end;
850     goto next_opt;
851
852 opt(group_protection):
853     longtemp = group_prot;
854     goto protection;
855
856 opt(ignore_line_marks):
857     fcb -> attr(atr_v_app_comma) = ^(specified & bitval(0));
858     goto next_opt;
859
860 opt(indexed):
861     if specified & bitval(0)
862         then do;
863         if fcb -> attr(atr_v_output) & ^fcb -> attr(atr_v_app)
864             then do;
865                 error_code = pli$creindex;
866                 goto opt_error;
867                 end;
868         fcb -> attr(atr_v_indexed) = true;
869         fcb -> fab$b_org = fab$c_idx;
870         end;
871     else do;
872         if fcb -> attr(atr_v_keyed) &
873             ^fcb -> fab$b_fac(fab$v_bio)
874         then fcb -> fab$b_org = fab$c_rel;
875         else fcb -> fab$b_org = fab$c_seq;
876         end;
877     goto next_opt;
878
879 opt(indexed_fill):
880     fcb -> rab$l_rop(rab$v_loa) = specified & bitval(0);
```



```
      goto next_opt;

opt(index_number):
  if specified
  then do;
    if longval(0) > 255
    then do;
      error_code = pli$_invindnum;
      goto opt_error;
    end;
    fcb -> rab$b_krf = byte;
  end;
  else fcb -> rab$b_krf = 0;
  goto next_opt;

opt(max_record_number):
  if specified
  then fcb -> fab$l_mrn = longval(0);
  else fcb -> fab$l_mrn = 0;
  goto next_opt;

opt(max_record_size):
  wordtemp = 0;
  bytetemp = fcb -> fab$b_fsz;
  if fcb -> fab$b_org = fab$c_rel
  then buflen = 480 - wordtemp;
  else do;
    if fcb -> fab$b_rfm = fab$c_fix
    then buflen = 512;
    else buflen = 510 - wordtemp;
  end;
  if specified
  then do;
    if longval(0) < 0 : longval(0) > 32767
    | (fcb -> fab$b_org = fab$c_rel &
      longval(0) > 16383)
    then do;
      error_code = pli$_invmaxrec;
      goto opt_error;
    end;
    fcb -> fab$w_mrs = word;
  end;
  else fcb -> fab$w_mrs = buflen;
  buflen = max(buflen, fcb -> fab$w_mrs);
  goto next_opt;

opt(multiblock_count):
  if specified
  then do;
    if longval(0) < 0 : longval(0) > 127
    then do;
      error_code = pli$_invmltblk;
      goto opt_error;
    end;
  end;
end;
```

```
943                                     end;
944         fcb -> rab$b_mbc = byte;
945     end;
946     else fcb -> rab$b_mbc = 0;
947     goto next_opt;
948
949
950 opt(multibuffer count):
951     if specified
952     then do;
953         if longval(0) < 0 | longval(0) > 127
954         then do;
955             error_code = pli$_invmltbuf;
956             goto opt_error;
957         end;
958         fcb -> rab$b_mbf = byte;
959     end;
960     else fcb -> rab$b_mbf = 0;
961     goto next_opt;
962
963
964 opt(no_share):
965     fcb -> fab$b_shr(fab$v_nil) = specified & bitval(0);
966     goto next_opt;
967
968
969 opt(owner_group):
970     if specified
971     then do;
972         if longval(0) < 0 | longval(0) > 255
973         then do;
974             error_code = pli$_invowngrp;
975             goto opt_error;
976         end;
977         own_group = word;
978         opn.status(uic) = true;
979     end;
980     goto next_opt;
981
982
983 opt(owner_member):
984     if specified
985     then do;
986         if longval(0) < 0 | longval(0) > 255
987         then do;
988             error_code = pli$_invownmem;
989             goto opt_error;
990         end;
991         own_mem = word;
992         opn.status(uic) = true;
993     end;
994     goto next_opt;
995
996
997 opt(owner_protection):
998     longtemp = owner_prot;
999     goto protection;
```

```
1000
1001
1002 opt(printer):
1003     if specified & bitval(0)
1004         then do;
1005             if fcb -> attr(atr_v_stream) !
1006                 fcb -> fab$b_rfm = fab$c_fix !
1007                 fcb -> fab$b_rat(fab$v_cr) !
1008                 fcb -> fab$b_fac(fab$v_bio)
1009             then do;
1010                 error_code = plis_conprtfm;
1011                 goto opt_error;
1012             end;
1013             fcb -> fab$b_rat(fab$v_prn) = true;
1014             fcb -> fab$b_rfm = fab$c_vfc;
1015             end;
1016         else fcb -> fab$b_rat(fab$v_cr) = ^ (fcb -> attr(atr_v_print) !
1017             carriage_specified_false);
1018     goto next_opt;
1019
1020
1021 opt(read_ahead):
1022     fcb -> rab$l_rop(rab$v_rah) = true;
1023     if specified
1024         then fcb -> rab$l_rop(rab$v_rah) = bitval(0);
1025     goto next_opt;
1026
1027
1028 opt(read_check):
1029     fcb -> fab$l_fop(fab$v_rck) = specified & bitval(0);
1030     goto next_opt;
1031
1032
1033 opt(record_id_access):
1034     if specified & bitval(0) & fcb -> fab$b_fac(fab$v_bio)
1035         then do;
1036             error_code = plis_conblokio;
1037             goto opt_error;
1038         end;
1039     fcb -> attr(atr_v_recidacc) = specified & bitval(0);
1040     goto next_opt;
1041
1042
1043 opt(retrieval_pointers):
1044     if specified
1045         then do;
1046             if longval(0) > 127 ! longval(0) < -1
1047                 then do;
1048                     error_code = plis_invrtvptr;
1049                     goto opt_error;
1050                 end;
1051             if longval(0) = -1
1052                 then longval(0) = 255;
1053             fcb -> fab$b_rtv = byte;
1054             end;
1055         else fcb -> fab$b_rtv = 0;
1056     goto next_opt;
```



```
1057
1058
1059 opt(rewind_close):
1060     fcb -> fab$l_fop(fab$V_rwc) = specified & bitval(0);
1061     goto next_opt;
1062
1063
1064 opt(rewind_open):
1065     fcb -> fab$l_fop(fab$V_rwo) = specified & bitval(0);
1066     goto next_opt;
1067
1068
1069 opt(scalarvarying):
1070     fcb -> attr(atr_v_scalvar) = specified & bitval(0);
1071     goto next_opt;
1072
1073
1074 opt(shared_read):
1075     if specified & bitval(0)
1076     then do;
1077         if fcb -> fab$b_shr(fab$V_nil)
1078         then do;
1079             error_code = pli$_noshare;
1080             goto opt_error;
1081             end;
1082             fcb -> fab$b_shr(fab$V_get) = true;
1083             fcb -> fab$b_shr(fab$V_upi) = true;
1084             end;
1085         else fcb -> fab$b_shr(fab$V_get) = false;
1086         goto next_opt;
1087
1088
1089 opt(shared_write):
1090     if specified & bitval(0)
1091     then do;
1092         if fcb -> fab$b_shr(fab$V_nil)
1093         then do;
1094             error_code = pli$_noshare;
1095             goto opt_error;
1096             end;
1097             fcb -> fab$b_shr(fab$V_put) = true;
1098             fcb -> fab$b_shr(fab$V_get) = true;
1099             fcb -> fab$b_shr(fab$V_del) = true;
1100             fcb -> fab$b_shr(fab$V_upd) = true;
1101             fcb -> fab$b_shr(fab$V_upi) = true;
1102             end;
1103         else do;
1104             fcb -> fab$b_shr(fab$V_put) = false;
1105             fcb -> fab$b_shr(fab$V_del) = false;
1106             fcb -> fab$b_shr(fab$V_upd) = false;
1107             end;
1108         goto next_opt;
1109
1110
1111 opt(spool):
1112     fcb -> fab$l_fop(fab$V_spl) = specified & bitval(0);
1113     goto next_opt;
```

```
1114  
1115  
1116  
1117 opt(supersede):  
1118     if specified & bitval(0)  
1119     then do;  
1120         if fcb -> attr(atr_v_app)  
1121         then do;  
1122             error_code = plis_conappsup;  
1123             goto opt_error;  
1124             end;  
1125         fcb -> fab$l_fop(fab$sv_mxv) = false;  
1126         fcb -> fab$l_fop(fab$sv_cif) = false;  
1127         fcb -> fab$l_fop(fab$sv_sup) = true;  
1128         fcb -> fab$l_fop(fab$sv_nef) = true;  
1129         fcb -> rab$l_rop(rab$sv_eof) = false;  
1130         end;  
1131     else do;  
1132         if ^fcb -> attr(atr_v_app)  
1133         then do;  
1134             fcb -> fab$l_fop(fab$sv_mxv) = false;  
1135             fcb -> fab$l_fop(fab$sv_cif) = false;  
1136             fcb -> fab$l_fop(fab$sv_sup) = false;  
1137             fcb -> fab$l_fop(fab$sv_nef) = false;  
1138             fcb -> rab$l_rop(rab$sv_eof) = false;  
1139             end;  
1140         goto next_opt;  
1141     end;  
1142  
1143 opt(system_protection):  
1144     longtemp = system_prot;  
1145     goto protection;  
1146  
1147  
1148 opt(temporary):  
1149     fcb -> fab$l_fop(fab$sv_tmp) = specified & bitval(0);  
1150     goto next_opt;  
1151  
1152  
1153 opt(truncate):  
1154     fcb -> fab$l_fop(fab$sv_tef) = specified & bitval(0);  
1155     goto next_opt;  
1156  
1157  
1158 opt(world_protection):  
1159     longtemp = world_prot;  
1160     goto protection;  
1161  
1162  
1163 opt(write_behind):  
1164     fcb -> rab$l_rop(rab$sv_wbh) = specified & bitval(0);  
1165     goto next_opt;  
1166  
1167  
1168 opt(write_check):  
1169     fcb -> fab$l_fop(fab$sv_wck) = specified & bitval(0);  
1170     goto next_opt;
```

PLISSENVIR
1-004

1171 2
1172 2

1 16
16-SEP-1984 02:29:38 VAX-11 PL/I X2.1-273 Page 21
6-SEP-1984 11:37:37 ISK\$VMSMASTER:[PLIRTL.SRC]PLIENVIR.PLI;1 (8)

```
1173 1      /* utility routines */
1174 1      protection:
1175 1          if specified
1176 1              then
1177 1                  if verify(addrval(0) -> stringtemp,'rwedRWED') ^= 0
1178 1                      then do;
1179 1                          error_code = plis_invprot;
1180 1                          goto opt_error;
1181 1                      end;
1182 1          if ^specified
1183 1              then do;
1184 1                  prot(longtemp + no_read) = true;
1185 1                  prot(longtemp + no_write) = true;
1186 1                  prot(longtemp + no_execute) = true;
1187 1                  prot(longtemp + no_delete) = true;
1188 1              end;
1189 1          else do;
1190 1              if (index(addrval(0) -> stringtemp,'r') = 0 &
1191 1                  index(addrval(0) -> stringtemp,'R') = 0)
1192 1                  then prot(longtemp + no_read) = true;
1193 1              if (index(addrval(0) -> stringtemp,'w') = 0 &
1194 1                  index(addrval(0) -> stringtemp,'W') = 0)
1195 1                  then prot(longtemp + no_write) = true;
1196 1              if (index(addrval(0) -> stringtemp,'e') = 0 &
1197 1                  index(addrval(0) -> stringtemp,'E') = 0)
1198 1                  then prot(longtemp + no_execute) = true;
1199 1              if (index(addrval(0) -> stringtemp,'d') = 0 &
1200 1                  index(addrval(0) -> stringtemp,'D') = 0)
1201 1                  then prot(longtemp + no_delete) = true;
1202 1              opn.status(protect) = true;
1203 1              end;
1204 1          goto next_opt;
1205 1
1206 1      /* bottom of loop */
1207 1
1208 1      next_opt:
1209 1          if specified
1210 1              then do;
1211 1                  if openv -> optbit.env_number = 0
1212 1                      then openv = addr(end_opt);
1213 1                  if declared_environment -> optbit.env_number = 0
1214 1                      then declared_environment = addr(end_opt);
1215 1                  if openv -> optbit.env_number =
1216 1                      declared_environment -> optbit.env_number
1217 1                      then do;
1218 1                          call get_opt_val(openv,0);
1219 1                          call get_opt_val(declared_environment,1);
1220 1                      end;
1221 1                  else do;
1222 1                      if openv -> optbit.env_number <
1223 1                          declared_environment -> optbit.env_number
1224 1                          then call get_opt_val(openv,0);
1225 1                      else call get_opt_val(declared_environment,0);
1226 1                  end;
1227 1              end;
1228 1          end;
```



```
1229      return(1);
1230
1231
1232      get_opt_val: procedure(optpt, valnum);
1233      /* parameter declarations */
1234      dcl optpt      pointer;
1235      dcl valnum      fixed bin(7);
1236
1237      next_specified_env_number = optpt -> optbit.env_number;
1238      if next_specified_env_number = 0 : next_specified_env_number = unused_envir_opt
1239      then do;
1240          next_specified_env_number = unused_envir_opt;
1241          return;
1242      end;
1243
1244      goto    opt_ttyp(env_type(next_specified_env_number));
1245
1246      opt_ttyp(bittyp):
1247          bitval(valnum) = optpt -> optbit.bitt;
1248          optpt = addr(optpt -> bitnext);
1249          if valnum = 1 & bitval(0) ^= bitval(1)
1250          then goto con_opt_exit;
1251          return;
1252
1253      opt_ttyp(longtyp):
1254          longval(valnum) = optpt -> long;
1255          optpt = addr(optpt -> longnext);
1256          if valnum = 1 & longval(0) ^= longval(1)
1257          then goto con_opt_exit;
1258          return;
1259
1260      opt_ttyp(quadtyp):
1261          quadval(valnum, 0) = optpt -> address -> qvalue(0);
1262          quadval(valnum, 1) = optpt -> address -> qvalue(1);
1263          optpt = addr(optpt -> addrnext);
1264          if valnum = 1 & (quadval(0, 0) ^= quadval(1, 0) :
1265              quadval(0, 1) ^= quadval(1, 1))
1266          then goto con_opt_exit;
1267          return;
1268
1269      opt_ttyp(stringtyp):
1270          addrval(valnum) = addr(optpt -> string);
1271          optpt = addr(optpt -> stringnext(optpt -> optstringnext.maxsize));
1272          if valnum = 1 & addrval(0) -> stringtemp ^=
1273              addrval(1) -> stringtemp
1274          then goto con_opt_exit;
1275          return;
1276
1277      opt_ttyp(addrtyp):
1278          addrval(valnum) = optpt -> address;
1279          optpt = addr(optpt -> addrnext);
1280          if valnum = 1 & addrval(0) ^= addrval(1)
1281          then goto con_opt_exit;
1282          return;
1283
1284      con_opt_exit:
1285          error_code = plis_conenvopt;
```

PLISSENVIR
1-004

L 16
16-SEP-1984 02:29:38
6-SEP-1984 11:37:37

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[PLIRTL.SRC]PLIENVIR.PLI;1 (9)

Page 24

1286	2	goto opt_error;
1287	2	
1288	2	end get_opt_val;
1289	1	
1290	1	end plissenvir;

COMMAND LINE

PLI/DEBUG=NONE/LIS=LIS\$:PLIENVIR/OBJ=OBJ\$:PLIENVIR MSRC\$:PLIENVIR+LIB\$:PL1RTSRC.TLB/LIB

0307 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

